Strings: An Overview

Defining a String

A string is an array of characters whose last character is \0. A typical string, such as Pacifique, is graphically represented as follows:

P	a	c	i	f	i	q	u	e	\0

The last character \0 is called the null-terminating character. For this reason, a string is said to be null-terminated. The string library ships with a lot of functions used to perform almost any type of operation on almost any kind of string. Used under different circumstances, the string functions also have different syntaxes.

The strings that are part of the C language are in the _str.h file. You should never include that library in your program. First of all, the iostream library includes the string library; which means by including the iostream library, you can perform any operation validated by the C string functions. Second, if you need to includce their library, use the string.h header file.

String Manipulation Functions

The C++ and its parent the C languages do not have a string data type. In C and C++, strings are created from arrays. Therefore, the C++ language relies on operations performed on the arrays of characters or pointers to char. The functions used for this purpose are numerous and you should know what they are used for.

As a reminder, thanks to the features of arrays of characters and the friendship between arrays of characters and strings, there are two main ways you can declare and initialize a string:

```
char *Thing = "Telephone";
char Major[] = "Computer Sciences";
```

The Length of a String

In many operations, you will want to know how many characters a string consists of. To find the number of characters of a string, use the strlen() function. Its syntax is:

```
int strlen(const char* Value);
```

The strlen() function takes one argument, which is the string you are considering. The function returns the number of characters of the string. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char *School = "Manchester United";
    int Length = strlen(School);
```



```
    cout << "The length of \"" << School
         << "\" is " << Length << " characters";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
The length of "Manchester United" is 17 characters

Press any key to continue...
```

The strcat() Function

If you want two strings, to append one to another, use the strcat() function. Its syntax is:

```
char *strcat(char *Destination, const char *Source);
```

The strcat() function takes two argument. The second argument, called the source string, is the string you want to add to the first string; this first string is referred to as the destination. Although the function takes two argument. It really ends up changing the destination string by appending the second string at the end of the first string. This could be used to add two strings. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char *Make = "Ford ";
    char *Model = "Explorer";

    cout << "Originally, Make = " << Make;

    strcat(Make, Model);

    cout << "\n\nAfter concatenating, Make = " << Make;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Originally, Make = Ford

After concatenating, Make = Ford Explorer

Press any key to continue...
```

The strncat() Function

Like the strcat() function, the strncat() function is used to append one string to another. The difference is that, while the strcat() considers all characters of the source string, the strncat() function allows you to specify the number of characters from the source string that you want to append to the destination

string. This means, if the source string has 12 characters, you can decide to append only a set number of its characters. The syntax is:

char* strncat(char* Destination, const char* Source, int Number);

Besides the same arguments as the strcat() function, the Number argument sets the number of characters considered from Source. To perform the concatenation, the compiler would count characters from left to right on the source string. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char *Make = "Ford ";
    char *Model = "Explorer";

    cout << "Originally, Make = " << Make;

    strncat(Make, Model, 3);

    cout << "\n\nAfter concatenating, Make = " << Make;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Originally, Make = Ford

After concatenating, Make = Ford Exp

Press any key to continue...
```

The strcpy Functtion

The strcpy function is used to copy one string into another string. In English, it is used to replace one string with another. The syntax of the strcpy function is:

char* strcpy(char* Destination, const char* Source);

The function takes two arguments. The first argument is the string that you are trying to replace. The second argument is the new string that you want to replace. There are two main scenarios suitable for the strcpy function: To replace an existing string or to initialize a string.

The following would initialize a string:

```
char CarName[20];

strcpy(CarName, "Toyota Camry");

cout << "Car Name: " << CarName;
```

If you have two string and copy one into another, both string would hold the same value:

```
//---------------------------------------------------------------------------
#include <iostream.h>
```



```
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char CarName1[] = "Ford Escort";
    char CarName2[] = "Toyota 4-Runner";

    cout << "The String Copy Operation";
    cout << "\nFirst Car: " << CarName1;
    cout << "\nSecond Car: " << CarName2;

    strcpy(CarName2, CarName1);

    cout << "\n\nAfter using strcpy()...";
    cout << "\nFirst Car: " << CarName1;
    cout << "\nSecond Car: " << CarName2;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
The String Copy Operation
First Car: Ford Escort
Second Car: Toyota 4-Runner

After using strcpy()...
First Car: Ford Escort
Second Car: Ford Escort

Press any key to continue...
```

The strncpy() Function

The strncpy() function works like the strcpy() function. As a difference, the strncpy() function allows you to specify the number of characters that the compiler would copy from the source string. Here is the syntax:

char* strncpy(char* Destination, const char* Source, int Number);

The Number argument specifies the number of characters that will be copied from the Source string. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char CarName1[] = "Ford Escort";
    char CarName2[] = "Toyota 4-Runner";

    cout << "The String Copy Operation";
    cout << "\nFirst Car: " << CarName1;
    cout << "\nSecond Car: " << CarName2;

    strncpy(CarName2, CarName1, 8);

    cout << "\n\nAfter using strncpy() for 8 characters";
    cout << "\nFirst Car: " << CarName1;
```

```
    cout << "\nSecond Car: " << CarName2;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
The String Copy Operation
First Car: Ford Escort
Second Car: Toyota 4-Runner

After using strncpy() for 8 characters
First Car: Ford Escort
Second Car: Ford Esc-Runner

Press any key to continue...
```

The strdup() Function

The strdup() function is used to make a copy of create a duplicate of that string. Its syntax is:

char* strdup(const char *S);

This function takes as an argument the string you want to duplication and returns the duplicated string.

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char *FirstName1 = "Charles";
    char *FirstName2 = "Andy";
    char *LastName1 = "Stanley";
    char *LastName2;

    LastName2 = strdup(LastName1);

    cout << "Father: " << FirstName1 << ' ' << LastName1 << endl;
    cout << "Son:    " << FirstName2 << ' ' << LastName2;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Father: Charles Stanley
Son:    Andy Stanley

Press any key to continue...
```

Comparing Strings

Various routines are available for strings comparison. The C-String library is equipped with functions that perform the comparisons by characters. Alternatively, the VCL also provides its own set of functions to perform these comparisons on null-terminated strings.

The strcmp() Function

The strcmp() function compares two strings and returns an integer as a result of its comparison. Its syntax is:

int strcmp(const char* S1, const char* S2);

This function takes two strings, S1 and S2 and compares them. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal
- A positive value if S1 is greater than S2

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char *FirstName1 = "Andy";
    char *FirstName2 = "Charles";
    char *LastName1  = "Stanley";
    char *LastName2  = "Stanley";

    int Value1 = strcmp(FirstName1, FirstName2);
    int Value2 = strcmp(FirstName2, FirstName1);
    int Value3 = strcmp(LastName1, LastName2);

    cout << "The result of comparing " << FirstName1
         << " and " << FirstName2 << " is\t" << Value1 << endl;
    cout << "The result of comparing " << FirstName2
         << " and " << FirstName1 << " is\t" << Value2 << endl;
    cout << "The result of comparing " << LastName1
         << " and " << LastName2 << " is\t" << Value3;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
The result of comparing Andy and Charles is     -2
The result of comparing Charles and Andy is     2
The result of comparing Stanley and Stanley is  0

Press any key to continue...
```

The strncmp() Function

The strncmp() function compares two strings using a specified number of characters and returns an integer as a result of its findings. Its syntax is:

int strncmp(const char* S1, const char* S2, int Number);

this function takes three arguments, the first two arguments are the strings that need to be compared. The 3rd argument specifies the number of characters considered for the comparison. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal

- A positive value if S1 is greater than S2

The stricmp() Function

The stricmp() function compares two strings without regard to their case; in other words, this function does not take into consideration if there is a mix of uppercase and lowercase in the strings. The syntax of the function is:

int stricmp(const char* S1, const char* S2);

This function takes two strings, S1 and S2 and compares them. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal
- A positive value if S1 is greater than S2

The strnicmp() Function

The strnicmp() function compares two strings without regard to their case but considering only a specified number of characters. The syntax of the function is:

int strnicmp(const char* S1, const char* S2, int n);

This function takes two strings, S1 and S2 and compares them. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal
- A positive value if S1 is greater than S2

The AnsiStrComp() Function

The AnsiStrComp() function is used to compare two strings with regard to case sensitivity. This function, when performed on dates and currency values, considers the Regional Settings of your computer. Its syntax is:

int __fastcall AnsiStrComp(char * S1, char * S2);

This function takes two strings, S1 and S2 and compares them. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal
- A positive value if S1 is greater than S2

Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char State1[] = "North Dakota";
    char State2[] = "South Dakota";

    cout << "State 1: " << State1 << "\n";
    cout << "State 2: " << State2 << "\n";
```



```
    int Result = AnsiStrComp(State1, State2);
    if( Result < 0 )
        cout << State1 << " is less than "
             << State2 << " by string comparison";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
State 1: North Dakota
State 2: South Dakota
North Dakota is less than South Dakota by string comparison

Press any key to continue...
```

The AnsiStrIComp() Function

The AnsiStrIComp() function compares two strings without regard to case sensitivity. This comparison is controlled by the Regional Settings of your computer. Its syntax is:

int __fastcall AnsiStrIComp(char * S1, char * S2);

This function takes two strings, S1 and S2 and compares them. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal
- A positive value if S1 is greater than S2

Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Word1[] = "resume";
    char Word2[] = "Resume";

    cout << "Word 1: " << Word1 << "\n";
    cout << "Word 2: " << Word2 << "\n";

    int Result = AnsiStrIComp(Word1, Word2);

    if( Result == 0 )
        cout << Word1 << " and " << Word2 << " are the same";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Word 1: resume
Word 2: Resume
resume and Resume are the same
```

```
Press any key to continue...
```

The AnsiStrLComp() Function

To perform string comparison using a set number of charaters, you can use the AnsiStrLComp() function. Like the AnsiStrComp() function, the AnsiStrLComp() function compares two strings with regard to case sensitivity and the Regional Settings of the computer. This time, the comparison is performed on a number of characters. Its syntax is:

int __fastcall AnsiStrLComp(char * S1, char * S2, Cardinal MaxLen);

This function compares strings S1 and S2 for a mxmum of MaxLen characters. It returns

- A negative value if S1 is less than S2
- 0 if S1 and S2 are equal
- A positive value if S1 is greater than S2

The implementation of the following example compares the 1st 6 characters of two strings:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Girl1[] = "Marcelle";
    char Girl2[] = "Marceline";

    cout << "Student 1: " << Girl1 << "\n";
    cout << "Student 2: " << Girl2 << "\n";

    int Result = AnsiStrLComp(Girl1, Girl2, 6);

    if( Result == 0 )
        cout << "If " << Girl1 << " and " << Girl2 << " were boys, "
             << "they would have the same name";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

The CompareMem() Function

The CompareMem() function is used to compare memory space of tow stored values using their addresses. Its syntax is:

bool __fastcall CompareMem(void * Mem1, void * Mem2, int Length);

Normally, this function is not solely intended for strings, but as you can see, it takes two general pointers and an integer as arguments. This function compares the contents of the Mem1 and the Mem2 variables for a length of Length bytes. When used with strings, it would be written as:

CompareMem(String1, String2, Length);

Therefore, it would compare String1 and String2 for Length characters

(remember that each character occupies a byte). If String1 is the same as String2, the function would return true; otherwise, it would return false. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char LastName1[] = "Martin";
    char LastName2[] = "Martyn";

    cout << "Last Name 1: " << LastName1 << "\n";
    cout << "Last Name 2: " << LastName2 << "\n";

    bool Result = CompareMem(LastName1, LastName2, 3);

    if( Result == True )
        cout << "Both boys have the same last name";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This version would compare the first 3 characters of Martin and Martyn. The result of this comparison would produce:

```
Last Name 1: Martin
Last Name 2: Martyn
Both boys have the same last name

Press any key to continue...
```

If you change the 3rd argument of the function to 5, the result would be different.

Working With Individual Characters

The strchr() Function

The strchr() function looks for the first occurrence of a certain character in a string. Its syntax is:

char* strchr(const char* S, char c);

This function takes two arguments. The second argument specifies what character to look for in the first argument which is a string. If the character c appears in the string S, the function would return a new string whose value starts at the first occurrence of c in S. If the character c does not appear in the string S, then the function would return NULL.

The strrchr() Functionn

The strrchr() function examines a string starting at the end (right side) of the string and looks for the first occurrence of a certain character. Its syntax is:

char* strrchr(const char* S, char c);

The first argument is the string that needs to be examined. The function will scan the string S from right to left. Once it finds the first appearance of the character c in the string, it would return a new string whose value starts at that first occurrence. If the character c does not appear in the string S, then the function would return NULL.

Working With Sub-Strings

The strstr() Function

The strstr() function looks for the first occurrence of a sub-string in another string and returns a new string as the remaining string. Its syntax is:

char* strstr(const char* Main, const char *Sub);

The first argument of the function is the main string that would be examined. The function would look for the second argument, the Sub string appearance in the main string. If the Sub string is part of the Main string, then the function would return a string whose value starts at the first appearance of Sub and make it a new string. If Sub is not part of the Main string, the function would return a NULL value.

Working With Character Cases

The strlwr() Function

The strlwr() function is used to convert a string to lowercase. Its syntax is:

char *strlwr(const char *S);

This function takes, as argument, the string that needs to be converted. During conversion, if a Latin character were in uppercase, it would be converted to lowercase. Otherwise, it would be stay “as if”. This means any symbol that is not a readable character would not be converted.

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{

    char CustomerAddress[] = "4812 LOCKWOOD Drive #F04";
    cout << "Customer Address: " << CustomerAddress << endl;

    char *ShippingAddress = strlwr(CustomerAddress);
    cout << "Shipping Address: " << ShippingAddress << endl;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Customer Address: 4812 LOCKWOOD Drive #F04
Shipping Address: 4812 lockwood drive #f04
```



```
Press any key to continue...
```

The AnsiStrLower() Function

The AnsiStrLower() function considers the Regional Settings and converts the characters of a string to lowercase. Its syntax is:

char * __fastcall AnsiStrLower(char * String);

This function examines each character of the String argument. If an alphabetical character is in UPPERCASE, then the function converts it to lowercase. All other letters and symbols keep their case. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Clause[] = "Maximum Characters = 1.";

    cout << "Clause: " << Clause << "\n";
    char *Converted = AnsiStrLower(Clause);

    cout << "\nAfter conversion,\nClause: " << Converted;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Clause: Maximum Characters = 1.

After conversion,
Clause: maximum characters = 1.

Press any key to continue...
```

The strupr() Function

The strupr() function is used to convert a string to uppercase. Its syntax is:

char *strupr(const char *S);

Each lowercase character in the function’s argument, S, would be converted to uppercase. Any character or symbol that is not in lowercase would not be changed.

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Drink[] = "100% Apple Juice";
    char *SayItLoud;
```

```
    cout << "What is that drink? " << Drink << endl;

    SayItLoud = strupr(Drink);
    cout << "Say it loud: " << SayItLoud << endl;

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
What is that drink? 100% Apple Juice
Say it loud: 100% APPLE JUICE

Press any key to continue...
```

The AnsiStrUpper() Function

The AnsiStrLower() function considers the Regional Settings and converts the characters of a string to uppercase. Its syntax is:

char * __fastcall AnsiStrUpper(char * String);

This function examines each character of the String argument. If an alphabetical character is in lowercase, then the function converts it to UPPDERCASE. All other letters and symbols keep their case. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <vcl.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Spreadsheet[] = "Application = Lotus 1-2-3";

    cout << "Spreadsheet: " << Spreadsheet << "\n";

    char *Application = AnsiStrLower(Spreadsheet);

    cout << "\nAfter conversion,\nApplication: " << Application;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Spreadsheet: Application = Lotus 1-2-3

After conversion,
Application: application = lotus 1-2-3

Press any key to continue...
```

Formatting Strings

The sprintf() Function

The sprintf() function is used to format data and specify how it should display.

Its syntax is:

int sprintf(char* Buffer, const char* S, Arguments...);

This function takes at least two arguments and could take more. The first argument is a null-terminated string that could display one or a few words and the formula to use when displaying the second or more argument. To display a value as part of the Buffer, type a double-quote followed by the string if any, followed by the % sign, follow by one of the following characters:

Character	Used to Display	Character	Used to Display
s	A string	f	Floating-point value
c	A character	d	An integer

Applying Strings

When using an object (class or structure), you can use the strcpy() function to initialize the strings. Consider the following class with variable strings:

```
//---------------------------------------------------------------------------
#ifndef CustomerH
#define CustomerH
//---------------------------------------------------------------------------
class TCustomer
{
public:
    __fastcall TCustomer();
    __fastcall ~TCustomer();
    void __fastcall RequestTheName();
    void __fastcall RequestTheAddress();
    void __fastcall DisplayName();
    void __fastcall DisplayAddress();
private:
    char FirstName[20];
    char LastName[20];
    char Address[50];
    char City[20];
    char State[3];
    long ZIPCode;
};
//---------------------------------------------------------------------------
#endif
```

Using the strcpy() function, you can initialize the strings as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

#include "Customer.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
__fastcall TCustomer::TCustomer()
{
    strcpy(FirstName, "Christine");
    strcpy(LastName, "Ndolo");
    strcpy(Address, "12044 Locker Avenue");
    strcpy(City, "Washington");
    strcpy(State, "DC");
    ZIPCode = 20012;
}
//---------------------------------------------------------------------------
```

```
__fastcall TCustomer::~TCustomer()
{
    //TODO: Add your source code here
}
//---------------------------------------------------------------------------
void __fastcall TCustomer::DisplayName()
{
    cout << "\nFull Name: " << FirstName << " " << LastName;
}
//---------------------------------------------------------------------------
void __fastcall TCustomer::DisplayAddress()
{
    cout << "\nAddress: " << Address;
    cout << "\nCity: " << City << ", " << State << " " << ZIPCode;
}
//---------------------------------------------------------------------------
```

Once the strings have been initialized and the object can be used, you can call the object from a necessary function:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
#include "Customer.h"
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    TCustomer Person;

    cout << "Customer Information";
    Person.DisplayName();
    Person.DisplayAddress();

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

This would produce:

```
Customer Information
Full Name: Christine Ndolo
Address: 12044 Locker Avenue
City: Washington, DC 20012

Press any key to continue...
```

The strcpy() function is also usually used to initialize access methods of an object. For example, you can declare an object's method such as:

```
void __fastcall setMakeModel(char m[]);
```

Since you would need to initialize the corresponding member variable by assigning the method's argument to the member variable, you can let the strcpy() function do that. The object would be declared as follows:

```
//---------------------------------------------------------------------------
#ifndef CarH
#define CarH
//---------------------------------------------------------------------------
enum TCarType { ctEconomy = 1, ctCompact, ctStandard,
                ctFullSize, ctMiniVan, ctSUV };
//---------------------------------------------------------------------------
class TCar
{
```



```
public:
    __fastcall TCar();
    __fastcall ~TCar();
    void __fastcall setMakeModel(char m[]);
    char* __fastcall getMakeModel();
    void __fastcall setCarYear(int y) { CarYear = y; }
    int __fastcall getCarYear() { return CarYear; }
    void __fastcall setCategory(int c) { Category = c; }
    int __fastcall getCategory() { return Category; }
private:
    char MakeModel[25];
    int CarYear;
    int Category;
};
//---------------------------------------------------------------------------
#endif
```

Therefore, you can use the strcpy() function to assign the argument to the right member variable to initialize it:

```
//---------------------------------------------------------------------------

#pragma hdrstop

#include "Car.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
__fastcall TCar::TCar()
{
    //TODO: Add your source code here
}
//---------------------------------------------------------------------------
__fastcall TCar::~TCar()
{
    //TODO: Add your source code here
}
//---------------------------------------------------------------------------
void __fastcall TCar::setMakeModel(char m[])
{
    strcpy(MakeModel, m);
}
//---------------------------------------------------------------------------
char* __fastcall TCar::getMakeModel()
{
    return MakeModel;
}
//---------------------------------------------------------------------------
```

Using a combination of the strcpy() function and array members, we can process an order for our car rental program. In the header file, we would declare arrays of characters. We would also declare methods that would return arrays of characters:

```
//---------------------------------------------------------------------------
#ifndef OrderH
#define OrderH
//---------------------------------------------------------------------------
#include "Customer.h"
#include "Car.h"
#include "RentDate.h"
//---------------------------------------------------------------------------
class TInvoice
{
public:
    __fastcall TInvoice();
    __fastcall ~TInvoice();
    void __fastcall RequestTheCar();
    void __fastcall CalculatePrice();
```

```
    void __fastcall setMileage(long g) { Mileage = g; }
    long __fastcall getMileage() { return Mileage; }
    void __fastcall setTankLevel(char v[]);
    char* __fastcall getTankLevel();
    void __fastcall setCarCondition(char c[]);
    char* __fastcall getCarCondition();
    double __fastcall getRate() { return Rate; }
    int __fastcall getNumberOfDays() { return NumberOfDays; }
    void __fastcall setDate(int m, int d, int y);
    void __fastcall RequestDate();
    void __fastcall ProcessOrder();
    TCustomer Cust; TCar Car;
    void __fastcall setStartDate(int m, int d, int y);
    void __fastcall setEndDate(int m, int d, int y);
    void __fastcall setRate();
    void __fastcall ExamineTheCar();
    void __fastcall ShowOrder();
private:
    int CarType;
    long Mileage;
    char TankLevel[10];
    char CarCondition[20];
    int NumberOfDays;
    double Rate;
    TRentDate StartDate;
    TRentDate EndDate;
    double OneDayRate,  // If renting for less than 5 days including week-end
           WeekDay, // If renting for at least 5 days, regardless of the days
           WeekEnd ; // If renting for more than two days from Friday to Monday
    double DayRate; // Rate applied based on the number of days
};
//---------------------------------------------------------------------------
#endif
```

When implementing the order processing, we would use the strcpy() function to initialize the array members. We would also use other techniques to pass strings to member methods:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

#include "Order.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
__fastcall TInvoice::TInvoice()
{
}
//---------------------------------------------------------------------------
__fastcall TInvoice::~TInvoice()
{
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::setTankLevel(char v[])
{
    strcpy(TankLevel, v);
}
//---------------------------------------------------------------------------
char* __fastcall TInvoice::getTankLevel()
{
    return TankLevel;
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::setCarCondition(char c[])
{
    strcpy(CarCondition, c);
}
//---------------------------------------------------------------------------
char* __fastcall TInvoice::getCarCondition()
```



```
{
    return CarCondition;
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::ProcessOrder()
{
    cout << "Enter Customer Information\n";
    Cust.RequestTheName();
    Cust.RequestTheAddress();

    cout << "\nCar Selection\n";
    RequestTheCar();
    CalculatePrice();
    ExamineTheCar();
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::RequestTheCar()
{
    int ModelSelected;

    // Request the type of car that the customer desires
    cout << "What type of car would you like to rent?";
    do {
        cout << "\n1 - Economy | 2 - Compact | 3 - Standard"
             << "\n4 - Full Size | 5 - Mini Van | 6 - Sports Utility";
        cout << "\nYour Choice: ";
        cin >> CarType;
        if( CarType < 1 || CarType > 6 )
            cout << "\nPlease type a number between 1 and 6";
    } while(CarType < 1 || CarType > 6);

    // The customer has selected a type
    // This means the CarType variable has been set and can be used
    // Depending on the type selected, ask the customer to select
    // a specific model of that category
    // Once the customer has selected a model, pass the name of the model
    // to the parent object so that name can be used as needed
    switch(CarType)
    {
    case ctEconomy:
        cout << "\nFor the Economy type, we have:"
             << "\n1 - Daewoo Lanos | 2 - Cheverolet Metro";
        cout << "\nWhich one would you prefer? ";
        cin >> ModelSelected;

        if(ModelSelected == 1)
            Car.setMakeModel("Daewoo Lanos");
        else
            Car.setMakeModel("Chevrolet Metro");
        break;

    case ctCompact:
        cout << "\nFor the Compact type, we have:"
             << "\n1 - Chevrolet Cavalier | 2 - Dogde Neon"
             << "\n3 - Nissan Sentra | 4 - Toyota Corolla";
        cout << "\nWhich one would you prefer? ";
        cin >> ModelSelected;

        if(ModelSelected == 1)
            Car.setMakeModel("Chevrolet Cavalier");
        else if(ModelSelected == 2)
            Car.setMakeModel("Dodge Neon");
        else if(ModelSelected == 3)
            Car.setMakeModel("Nissan Sentra");
        else
            Car.setMakeModel("Toyota Corrolla");
        break;

    case ctStandard:
        cout << "\nFor the Standard type, we have:"
             << "\n1 - Chevrolet Monte Carlo | 2 - Toyota Camri";
```

```
        cout << "\nWhich one would you prefer? ";
        cin >> ModelSelected;

        if(ModelSelected == 1)
            Car.setMakeModel("Chevrolet Monte Carlo");
        else
            Car.setMakeModel("Toyota Camri");
        break;

    case ctFullSize:
        cout << "\nFor the Full Size type, we have:"
             << "\n1 - Chrysler 300M | 2 - Buick Century
             << " | 3 - Infinity I30";
        cout << "\nWhich one would you prefer? ";
        cin >> ModelSelected;

        if(ModelSelected == 1)
            Car.setMakeModel("Chrysler 300M");
        else if(ModelSelected == 2)
            Car.setMakeModel("Buick Century");
        else
            Car.setMakeModel("Infinity I30");
        break;

    case ctMiniVan:
        cout << "\nFor the Mini-Van type, we have:"
             << "\n1 - Dodge Caravan | 2 - Dodge Caravan"
             << "\n3 - Pontiac Montana | 4 - Pontiac Montana
             << " | 5 - Chevrolet Astro Van";
        cout << "\nWhich one would you prefer? ";
        cin >> ModelSelected;

        if(ModelSelected == 1)
            Car.setMakeModel("Dodge Caravan");
        else if(ModelSelected == 2)
            Car.setMakeModel("Dodge Caravan");
        else if(ModelSelected == 3)
            Car.setMakeModel("Pontiac Montana");
        else
            Car.setMakeModel("Chevrolet Astro Van");
        break;

    case ctSUV:
        cout << "\nFor the Sport Utility type, we have:"
             << "\n1 - GMC Jimmy | 2 - Jeep Cherokee"
             << "\n3 - Chevrolet Blazer | 4 - Toyota Pathfinder";
        cout << "\nWhich one would you prefer? ";
        cin >> ModelSelected;

        if(ModelSelected == 1)
            Car.setMakeModel("GMC Jimmy");
        else if(ModelSelected == 2)
            Car.setMakeModel("Jeep Cherokee");
        else if(ModelSelected == 3)
            Car.setMakeModel("Chevrolet Blazer");
        else Car.setMakeModel("Toyota Pathfinder");
        break;
    }

    // Update the car type selected so that if any function wants it
    // a valid type would be avaiallable
    Car.setCategory(CarType);
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::setRate()
{
    // The CarType variable is now available because it was update
    // by the RequestTheCar method that was called previously
    // We can set the prices of rent depending on the model selected
    switch(CarType)
    {
```



```
    case ctEconomy:
        OneDayRate = 29.95;
        WeekDay = 24.95;
        WeekEnd = 19.95;
        break;

    case ctCompact:
        OneDayRate = 39.95;
        WeekDay = 34.95;
        WeekEnd = 25.95;
        break;

    case ctStandard:
        OneDayRate = 49.95;
        WeekDay = 38.95;
        WeekEnd = 32.95;
        break;

    case ctFullSize:
        OneDayRate = 69.95;
        WeekDay = 44.95;
        WeekEnd = 35.95;
        break;

    case ctMiniVan:
        OneDayRate = 89.95;
        WeekDay = 54.95;
        WeekEnd = 42.95;
        break;

    case ctSUV:
        OneDayRate = 89.95;
        WeekDay = 64.95;
        WeekEnd = 50.95;
        break;
    }
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::CalculatePrice()
{
    char WeekEndResponse;

    // Does the customer rent the car for the week-end?
    setRate();
    // Request the starting and ending dates the customer
    // wants to keep the car
    cout << "\nStart Renting Date\n";
    StartDate.RequestDate();
    cout << "\nEnd Renting Date\n";
    EndDate.RequestDate();
    // We can calculate the number of days the customer wants
    // to keep the car, based on the dates entered
    if( EndDate.getDay() <= StartDate.getDay() )
    {
        EndDate = StartDate;
        NumberOfDays = 1;
    }
    else
        NumberOfDays = EndDate.getDay() - StartDate.getDay();

    if( NumberOfDays < 5 )
    {
        // Find out if the customer qualifies for the week-end rate
        // The week-end rate applies only if the car is kept for
        // 1 to 3 days, that is Friday, Saturday, and Sunday
        if( NumberOfDays <= 3 )
        {
            // The clerk has the responsibility to check that the day(s)
            // that the customer selected fall(s) on a week-end.
            // This program WILL NOT check it
            // For this reason, the following question should be answered
```

```
            // by the clerk
            cout << "\nWe are having a special rate for the week-end."
                 << "\nIt includes Friday, Saturday, and Sunday";
            cout << "\nWill you use the car over the week-end(y=Yes/n=No)? ";
            cin >> WeekEndResponse;

            if( WeekEndResponse == 'y' || WeekEndResponse == 'Y' )
            {
                // If the customer will have the car over the week-end
                // apply the week-end discount to the rate
                DayRate = WeekEnd;
            }
            else
            {
                // Since the customer will not have the car over the week-end
                // Apply the regular daily rate
                DayRate = OneDayRate;
            }
        }
        else
        {
            // It appears that the customer will have the car for
            // at least 5 days. Even if the period of rent falls
            // in a week-end, since at least one day falls outside
            // of the week-end, the customer must pay the regular rate
            DayRate = OneDayRate;
        }
    }
    else
    {
        // Since the customer will have the car for at least 5 days
        // We have a special week rate.
        DayRate = WeekDay;
    }

    // We can now calculate the total rate based on the number of days
    // We know whether the customer can receive a discount and why
    Rate = NumberOfDays * DayRate;
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::ExamineTheCar()
{
    char Cond;
    int GasLevel;
    unsigned int YearOfCar;

    cout << "\nNow examining the car";
    do {
        cout << "\nType the year the car was made: ";
        cin >> YearOfCar;
    } while( YearOfCar >= 2002 );

    Car.setCarYear(YearOfCar);

    cout << "Rate the car's condition (e=Excellent/g=Good/d=Driveable): ";
    cin >> Cond;

    if( Cond == 'e' || Cond == 'E' )
        setCarCondition("Excellent");
    else if( Cond == 'g' || Cond == 'G' )
        setCarCondition("Good");
    else if( Cond == 'd' || Cond == 'D' )
        setCarCondition("Driveable");
    else
        setCarCondition("Can't Decide");

    cout << "Enter the car mileage: ";
    cin >> Mileage;

    do {
        cout << "Gas level in the tank"
```



```
             << "\n1 - Considered Empty"
             << "\n2 - 1/4 Full" << "\n3 - Half Full"
             << "\n4 - 3/4 Full" << "\n5 - Full Tank";
        cout << "\nEnter the gas level: ";
        cin >> GasLevel;
    } while(GasLevel < 1 || GasLevel > 5);

    switch(GasLevel)
    {
    case 1:
        setTankLevel("Empty");
        break;

    case 2:
        setTankLevel("1/4 Full");
        break;

    case 3:
        setTankLevel("Half Full");
        break;

    case 4:
        setTankLevel("3/4 Full");
        break;

    case 5:
        setTankLevel("Full Tank");
        break;
    }
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::setStartDate(int m, int d, int y)
{
    StartDate.setDate(m, d, y);
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::setEndDate(int m, int d, int y)
{
    EndDate.setDate(m, d, y);
}
//---------------------------------------------------------------------------
void __fastcall TInvoice::ShowOrder()
{
    cout << "\n===================================";
    cout << "\n-+-+-+- Bethesda Car Rental -+-+-+-";
    cout << "\n===================================";
    cout << "\n-------- Customer ---------";

    Cust.DisplayName();
    Cust.DisplayAddress();

    cout << "\n-------- Car Information ----------";
    cout << "\nModel:      " << Car.getMakeModel();
    cout << "\nYear:       " << Car.getCarYear();
    cout << "\nCondition:  " << getCarCondition();
    cout << "\nMileage:    " << getMileage();
    cout << "\nTank Level: " << getTankLevel();
    cout << "\nStart Date: ";

    StartDate.ShowDate();
    cout << "\nEnd Date: ";
    EndDate.ShowDate();
    cout << "\nNbr of Days: " << getNumberOfDays();
    cout << "\nTotal Price: $" << getRate();
    cout << "\n===================================\n";
}
//---------------------------------------------------------------------------
```

Now can can run the program. To test the program, from the main() function, we would declare a TInvoice class and call the appropriate methods:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
#include "Order.h"
//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    TInvoice Order;

    Order.ProcessOrder();
    system("cls");
    Order.ShowOrder();

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Here is an example of running the program:

Order Processing

```
Enter Customer Information
Enter First Name: Paul
Enter Last Name:  Benning
Address:  4405 Evening Drive #D12
City:     Takoma Park
State:    Maryland
ZIP Code: 20910

Car Selection
What type of car would you like to rent?
1 - Economy | 2 - Compact | 3 - Standard
4 - Full Size | 5 - Mini Van | 6 - Sports Utility
Your Choice: 2

For the Compact type, we have:
1 - Chevrolet Cavalier | 2 - Dogde Neon
3 - Nissan Sentra | 4 - Toyota Corolla
Which one would you prefer? 2

Start Renting Date
Month: 10
Day:   14
Year:  2002

End Renting Date
Month: 10
Day:   15
Year:  2002

We are having a special rate for the week-end.
It includes Friday, Saturday, and Sunday
Will you use the car over the week-end(y=Yes/n=No)? n

Now examining the car
Type the year the car was made: 2000
Rate the car's condition (e=Excellent/g=Good/d=Driveable): e
Enter the car mileage: 8744
Gas level in the tank
1 - Considered Empty
2 - 1/4 Full
3 - Half Full
4 - 3/4 Full
5 - Full Tank
Enter the gas level: 2
```


Invoice:

```
===================================
-+-+-+- Bethesda Car Rental -+-+-+-
===================================
-------- Customer ---------
Full Name: Paul Benning
Address: 4405 Evening Drive #D12
City: Takoma Park, Maryland 20910
-------- Car Information ----------
Model:      Dodge Neon
Year:       2000
Condition:  Excellent
Mileage:    8744
Tank Level: 1/4 Full
Start Date: 10/14/2002
End Date:   10/15/2002
Nbr of Days: 1
Total Price: $39.95
===================================

Press any key to continue...
```

Previous Next